# X WEB SCRAPING

*Maamoun GHNIMI*
*Wacel RAHAL*

# OUTLINE

01
Main Concepts

02
Main Existing Solutions

03
High Level Design

04
Tools and Development Phase

05
Executable Version

06
Summary

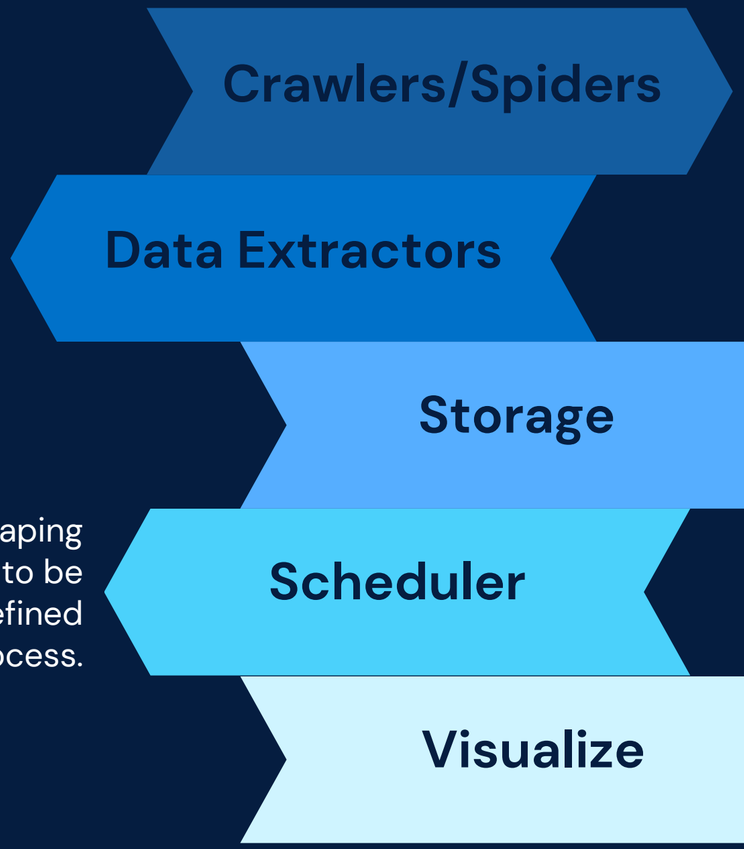
# Definition :

A web scraper is a sophisticated tool designed to navigate the vast expanse of the web, identifying and extracting data for various uses. At its core, a web scraper consists of several key components, each serving a distinct purpose in the data extraction process.

Web scraping is the process of automatically extracting data from a website. You use a program called a web scraper to access a web page, interpret the data, and extract what you need. The data is saved in a structured format such as an Excel file, JSON, or XML so that you can use it in spreadsheets or apps.

# MAIN CONCEPTS

# Components :




Crawlers, or spiders, are the first step in the web scraping process. These automated bots are programmed to surf the internet to find and retrieve web pages.

Once a crawler retrieves a web page, the data extractor takes over. This component is responsible for parsing the HTML, XML, or JSON content of the page to identify and extract the information of interest

The extracted data needs to be stored for further analysis or use. This component of the web scraper determines how and where data is saved.

The scheduler is the orchestrator of the scraping process. It manages the queue of URLs to be visited, prioritizing them based on predefined criteria to ensure an efficient scraping process.

The results of the analysis are visualized using charts, graphs to help understand the data.

# FUNCTIONAL FLOW OF WEB SCRAPING

The process of web scraping follows a logical sequence of steps, ensuring that data is collected systematically and efficiently.

| | |
|---|---|
| 1- URL Identification: | The first step involves identifying the URLs of the pages to scrape. This list can be generated dynamically or based on a predefined set of targets. |
| -2-Request Sending: | The scraper sends HTTP requests to the web servers hosting the target pages. Here, techniques like rotating user agents or using proxy servers might be employed to mimic human behavior and avoid detection. |
| -3- Data Parsing and Extraction: | Upon receiving the web page content, the scraper parses the HTML or other formats to locate the data of interest. Tools and libraries like Beautiful Soup for Python are commonly used for this purpose, allowing for easy navigation of the DOM tree. |
| -4-Data Storage and Analysis: | The extracted information is then saved in the chosen format. Subsequent analysis or processing can transform this data into valuable insights, such as trend analysis in social media posts or price monitoring across e-commerce platforms. |

# EXISTING SOLUTIONS

# EXISTING SOLUTIONS :

## Twitter User Information Scraping

**-Details:** Extracts user-related data such as usernames, display names, bio descriptions, locations, follower counts, and tweet statistics.

***Existing Products:***

**-Twint:** Command-line tool for scraping tweets, user profiles, followers list, and more without API limits.

**- Tweepy:** Python library for easy access to Twitter API, collecting user information with proper authentication.

**- Followerwonk:** Analyzes followers of Twitter accounts and provides detailed user information, especially for audience analysis.

## Twitter Tweet Scraping

**-Details:** Gathers tweets based on specific queries such as hashtags, keywords, date ranges, and geographical locations. Can also scrape retweets, likes, and replies.

***Existing Products:***

**- GetOldTweets3:** A Python script that allows for retrieving historical tweets before the limitations of Twitter's standard API.
**- ScrapeStorm:** AI-powered visual web scraping tool which can be configured to extract tweets displayed on user profiles or search results.
**- TweetScraper:** A Scrapy-based scraper that uses Twitter's Search API to download tweets without needing user authentication.

## Twitter Sentiment Analysis

**-Details:** Focuses on extracting tweets for sentiment analysis, which involves determining the sentiment or emotion behind tweets, categorizing them as positive, negative, or neutral.

***Existing Products:***

**- MonkeyLearn:** Offers a pre-trained sentiment analysis model that can be integrated with Twitter data scraping tools for automated sentiment classification.
**- MeaningCloud:** Provides text analytics services, including sentiment analysis, that can be used to analyze Twitter data once it has been scraped.
**- VaderSentiment:** A Python library specifically designed for sentiment analysis of social media texts, which can be applied to scraped Twitter data.

## ***ADVANTAGES :***

**Comprehensive Data Collection:** Tools like Twint and GetOldTweets3 allow for the extraction of a wide range of data, including historical tweets unavailable via the official API.

**User-Friendly Interfaces:** Tools with GUIs, such as ScrapeStorm, offer an easier entry point for users without technical expertise to perform web scraping tasks.

**No API Rate Limits:** Since some tools operate without requiring Twitter API access, they are not constrained by the rate limits, allowing for more extensive and rapid data collection.

## ***LIMITATIONS:***

**-Dependency on Twitter's Structure:** Tools that do not use the API, like Twint and TweetScraper, may break when Twitter updates its platform, requiring constant maintenance to stay functional.

**-Ethical and Legal Risks:** Using scraping tools can lead to violations of Twitter’s terms of service and potential legal issues, especially if the scraped data is not handled responsibly.

# HIGH LEVEL DESIGN

# Components :

**User Interface (UI):**

A front-end portal for users to input search parameters and initiate scraping tasks.

**Controller:**

Interprets user input from the UI and sends commands to the scraping engine.

**Authentication Module:**

Handles OAuth authentication for Twitter API access, managing tokens and user sessions.

**Scraping Engine:**

The core component that executes the scraping process, interfacing with Twitter.

**Data Processor:**

Cleanses and structures the raw data into a usable format.

# Components :

**Database:**

Stores the scraped and processed tweet data.

**Analysis Module:**

Provides basic analytical functions on the scraped data, such as counting occurrences of hashtags or keywords.

**Notification System:**

Alerts users to the status of scraping operations, including successes and errors.

# Data Flow :

(1)The User Interface collects input criteria from the user, such as search keywords, hashtags, or specific Twitter accounts.

(2)The Controller receives the input and instructs the Authentication Module to secure a session with Twitter.

(3)The Scraping Engine, once authenticated, uses the provided criteria to collect data from Twitter.

(4)Collected data is passed to the Data Processor, where it is cleaned and structured.

(5)Processed data is stored in the Database for persistence and further analysis.

(6)The Analysis Module accesses the data to generate insights, which can then be presented back to the user via the User Interface.

(7)Throughout the process, the Notification System keeps the user informed about the status of their scraping job.

# Exchanged Messages and Data:

Input from Users: Authentication credentials, scraping parameters (keywords, hashtags, etc.).

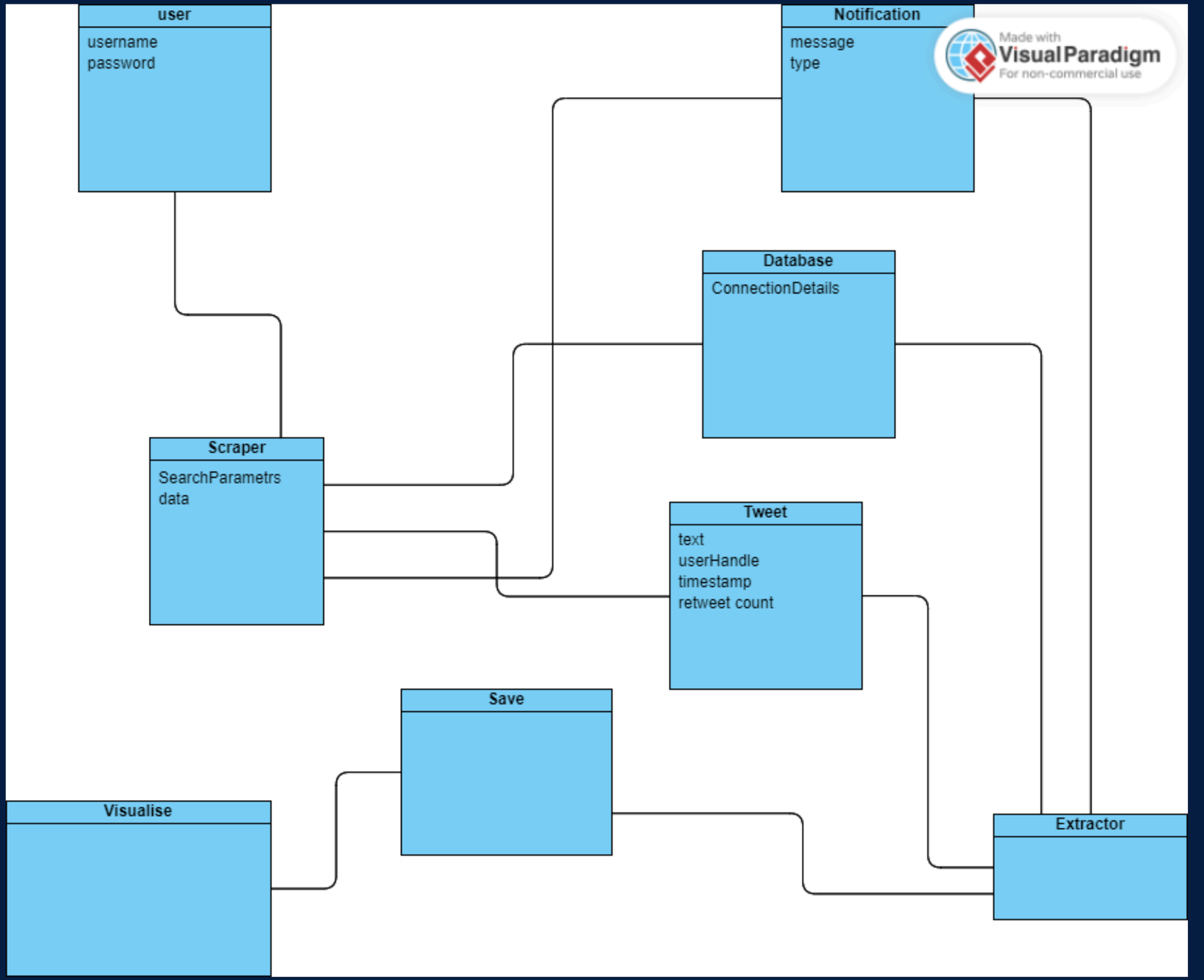
Requests to Twitter: API requests for data retrieval or HTML page requests for scraping.

Raw Data from Twitter: Tweet content, metadata, user information, etc.

Processed Data: Clean and structured datasets ready for analysis or reporting.

Analysis Results: Insights, statistics, and visualizations derived from the processed data.

Notifications: Status updates, error messages, and alerts sent to users.


user
username
password
Notification
message
type
Made with
VisualParadigm
For non-commercial use
Database
ConnectionDetails
Scraper
SearchParametrs
data
Tweet
text
userHandle
timestamp
retweet count
Save
Visualise
Extractor

# TOOLS AND DEVELOPMENT PHASES

# Python
# ( anacoda)

```python
import twint
import pandas as pd

def setup_twint_config(search_term, limit, output_filename):
    config = twint.Config()
    config.Search = search_term
    config.Limit = limit
    config.Store_csv = True
    config.Output = output_filename
    config.Lang = "en"
    return config

def scrape_tweets(search_term, limit=100, output_filename="output_tweets.csv"):
    config = setup_twint_config(search_term, limit, output_filename)
    try:
        twint.run.Search(config)
        print("Scraping completed successfully.")
    except Exception as e:
        print(f"An error occurred during scraping: {e}")

```

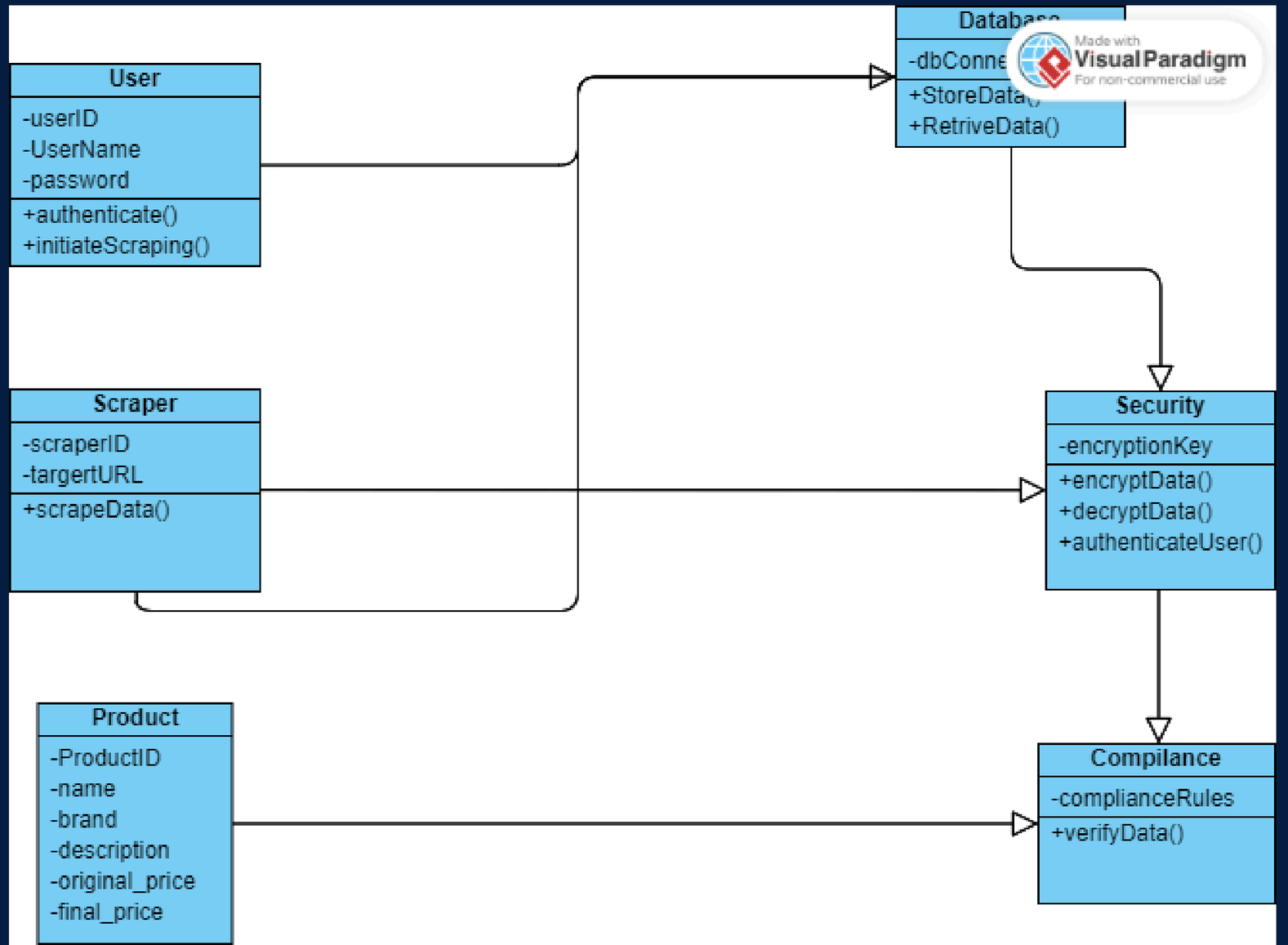
# Python
# ( Vs Code)

```python
def main():
    search_term = input("Enter the search term for scraping tweets: ")
    limit = int(input("How many tweets do you want to scrape? "))
    output_filename = input("Enter the output CSV filename: ")

    print(f"Starting to scrape tweets for '{search_term}'...")
    scrape_tweets(search_term, limit, output_filename)

    try:
        processed_tweets = pd.read_csv(output_filename)
        print("Tweets processed successfully. Here's a preview:")
        print(processed_tweets.head())
    except Exception as e:
        print(f"Failed to load or process tweets: {e}")

if __name__ == "__main__":
    main()
```

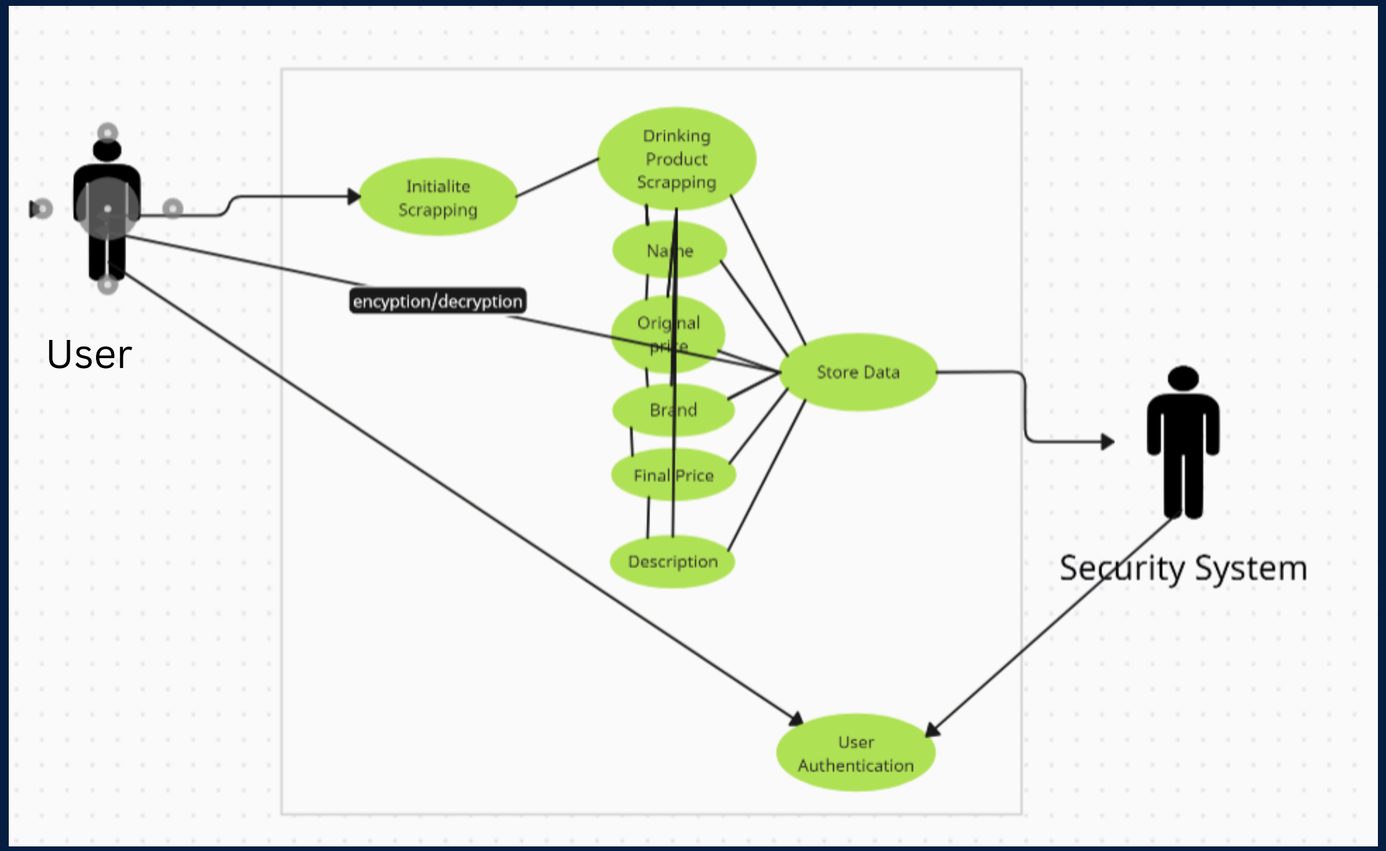
# NOTE

Just a quick reminder that I've shifted my project topic from Twitter web scraping to Monoprix web scraping due to an API problem. Looking forward to sharing my progress during the same presentation.

# Class Diagram:



# Use Case Diagram



# Python
# ( Vs Code)

(1)

```
import requests
from bs4 import BeautifulSoup
import xlwt
import warnings
from urllib3.exceptions import InsecureRequestWarning
```

```
# Suppress only the single InsecureRequestWarning from urllib3 needed to silence the warning.
warnings.simplefilter('ignore', InsecureRequestWarning)

def scrape_data_and_generate_excel(base_url):
    response = requests.get(base_url, verify=False)
    soup = BeautifulSoup(response.content, "html.parser")

    # Determine the number of pages
    pagination = soup.find("ul", class_="page-list clearfix text-xs-right")
    last_page = int(pagination.find_all("li")[-2].text) if pagination else 1

    data = []
    for page_number in range(1, last_page + 1):
        url = f"{base_url}?page={page_number}" if page_number > 1 else base_url
        response = requests.get(url, verify=False)
        soup = BeautifulSoup(response.content, "html.parser")

        product_titles = [elem.text.strip() for elem in soup.find_all("div", class_="h3 product-title")]
        product_marques = [elem.text.strip() for elem in soup.find_all("div", class_="div_manufacturer_name")]
        product_descriptions = [elem.text.strip() for elem in soup.find_all("div", class_="div_contenance")]
        original_prices = [elem.text.strip() for elem in soup.find_all("div", class_="regular-price")]
        final_prices = [elem.text.strip() for elem in soup.find_all("span", class_="price")]
```

# Python
# ( Vs Code)

(2)

```
        for i in range(len(product_titles)):
            data.append({
                "name": product_titles[i] if i < len(product_titles) else "",
                "marque": product_marques[i] if i < len(product_marques) else "",
                "description": product_descriptions[i] if i < len(product_descriptions) else "",
                "original_price": original_prices[i] if i < len(original_prices) else "",
                "final_price": final_prices[i] if i < len(final_prices) else "",
            })

        if last_page == 1:
            break

    file_name = "monoprix_boissons.xls"
    write_data_to_excel(data, file_name)
    return data, file_name

def write_data_to_excel(data, file_name):
    workbook = xlwt.Workbook(encoding="utf-8")
    sheet = workbook.add_sheet("Products")

    headers = ["name", "marque", "description", "original_price", "final_price"]
    for col, header in enumerate(headers):
        sheet.write(0, col, header)

    for row, item in enumerate(data, start=1):
        sheet.write(row, 0, item["name"])
        sheet.write(row, 1, item["marque"])
        sheet.write(row, 2, item["description"])
        sheet.write(row, 3, item["original_price"])
        sheet.write(row, 4, item["final_price"])

    workbook.save(file_name)
```

# Python
# ( Vs Code)

```python
def main():
    search_term = input("Enter the search term for scraping tweets: ")
    limit = int(input("How many tweets do you want to scrape? "))
    output_filename = input("Enter the output CSV filename: ")

    print(f"Starting to scrape tweets for '{search_term}'...")
    scrape_tweets(search_term, limit, output_filename)

    try:
        processed_tweets = pd.read_csv(output_filename)
        print("Tweets processed successfully. Here's a preview:")
        print(processed_tweets.head())
    except Exception as e:
        print(f"Failed to load or process tweets: {e}")

if __name__ == "__main__":
    main()
```

```python
if __name__ == "__main__":
    url_to_scrape = "https://courses.monoprix.tn/13-boissons"
    data, file_name = scrape_data_and_generate_excel(url_to_scra
    print(f"Data has been scraped and saved to {file_name}")
```

# Encryption

```python
from cryptography.fernet import Fernet

# Generate a key
key = Fernet.generate_key()

# Save the key to a file
with open('encryption_key.key', 'wb') as key_file:
    key_file.write(key)

# Create a Fernet symmetric key
cipher = Fernet(key)

# Read the contents of the clean output CSV file
input_file_path = r"C:\Users\ASUS\Desktop\Security__project\monoprix_boissons.csv"
with open(input_file_path, 'rb') as file:
    data = file.read()

# Encrypt the data
encrypted_data = cipher.encrypt(data)

# Write the encrypted data to a new file
output_file_path = r"C:\Users\ASUS\Desktop\Security__project\monoprix_boissons.csv"
with open(output_file_path, 'wb') as file:
    file.write(encrypted_data)

print("File encrypted successfully.")
```

# Decryption

```python
from cryptography.fernet import Fernet

# Load the key (replace 'key.txt' with the path to your key file)
with open('encryption_key.key', 'rb') as key_file:
    key = key_file.read()

# Create a Fernet symmetric key
cipher = Fernet(key)

# Read the contents of the encrypted file
input_file_path = r"C:\Users\ASUS\Desktop\Security__project\monoprix_boissons.csv"
with open(input_file_path, 'rb') as file:
    encrypted_data = file.read()

# Decrypt the data
decrypted_data = cipher.decrypt(encrypted_data)

# Write the decrypted data to a new file
output_file_path = r"C:\Users\ASUS\Desktop\Security__project\monoprix_boissons_dec.csv"
with open(output_file_path, 'wb') as file:
    file.write(decrypted_data)

print("File decrypted successfully.")
```